

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2002-055848

(43)Date of publication of application : 20.02.2002

(51)Int.Cl.

G06F 11/28

(21)Application number : 2000-245341

(71)Applicant : OMRON CORP

(22)Date of filing : 11.08.2000

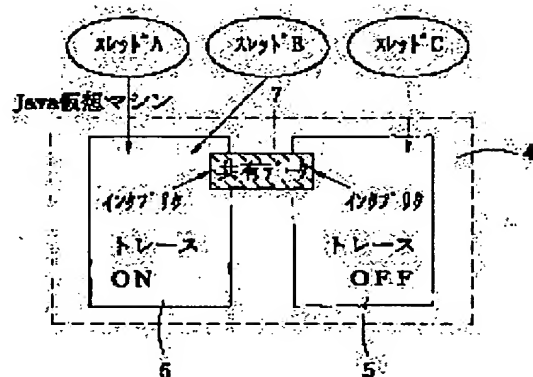
(72)Inventor : SAIKI HIDEAKI
HIRONO MITSUAKI
KONAKA YOSHIHARU
KADOWAKI MASANORI
MURAI KENICHI

(54) PROGRAM EXECUTION PROCEDURE AND STORAGE MEDIUM WITH THE PROGRAM EXECUTION PROCEDURE STORED THEREIN

(57)Abstract:

PROBLEM TO BE SOLVED: To provide a program execution procedure which enables easy and efficient debug of an application program in a program execution environment.

SOLUTION: The program execution environment constituted of a Java virtual machine 4 is provided with a normal interpreter 5 for interpreting a thread by a normal interpreter format and executing the thread (1) and a tracing interpreter 6 for tracing the thread by executing a program having a tracing function for each line or for each instruction and outputting the contents of a memory to the external (2). Both the interpreters 5, 6 can be mutually switched, the normal interpreter 5 can execute the thread and the tracing interpreter 6 can trace and debug the thread.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's
decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号
特開2002-55848
(P2002-55848A)

(43) 公開日 平成14年2月20日 (2002. 2. 20)

(51) Int.Cl. ⁷	識別記号	F I	テームコード* (参考)
G 0 6 F 11/28	3 1 0	G 0 6 F 11/28	3 1 0 A 5 B 0 4 2
	3 4 0		3 4 0 C

審査請求 未請求 請求項の数 2 O L (全 11 頁)

(21) 出願番号 特願2000-245341 (P2000-245341)

(22) 出願日 平成12年8月11日 (2000. 8. 11)

(71) 出願人 000002945

オムロン株式会社

京都市下京区塩小路通堀川東入南不動堂町
801番地

(72) 発明者 齋木 秀明

京都府京都市下京区西洞院木津屋橋通東入
ル オムロンソフトウェア株式会社内

(72) 発明者 廣野 光明

京都府京都市右京区花園土堂町10番地 オ
ムロン株式会社内

(74) 代理人 100094019

弁理士 中野 雅房

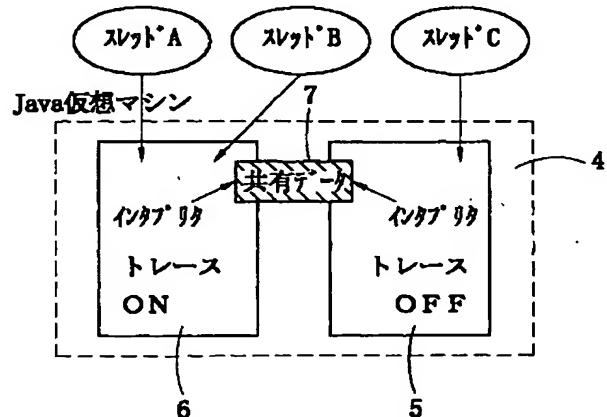
最終頁に続く

(54) 【発明の名称】 プログラム実行方式及び当該プログラム実行方式を格納した記憶媒体

(57) 【要約】

【課題】 プログラム実行環境において、アプリケーションプログラムを容易かつ作業効率よくデバッグできるプログラム実行方式を提供する。

【解決手段】 J a v a 仮想マシン 4 によって構成されたプログラム実行環境は、(1) 通常のインタプリタ形式でスレッドを解釈して実行する通常インタプリタ 5 と、(2) トレース機能を有してプログラムを 1 ラインや 1 命令ずつ実行し、トレースを行ってメモリの内容を外部に出力するトレース用インタプリタ 6 とを備えている。両インタプリタ 5、6 は切り替え可能となっており、通常インタプリタ 5 によりスレッドを実行でき、またトレース用インタプリタ 6 によりスレッドをトレースしてデバッグを行うことができる。



【特許請求の範囲】

【請求項1】 トレース機能を有する環境でプログラムを実行することができるプログラム実行環境と、トレース機能を有しない環境で実行することができるプログラム実行環境とを備え、前記両プログラム実行環境が選択的に実行可能となっていることを特徴とするプログラム実行方式。

【請求項2】 トレース機能を有する環境でプログラムを実行することができるプログラム実行環境と、トレース機能を有しない環境で実行することができるプログラム実行環境とを備え、両プログラム実行環境が選択的に実行可能となっているプログラム実行方式を格納した記憶媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】 本発明は、プログラム実行方式及び当該プログラム実行方式を格納した記憶媒体に関する。具体的には、本発明は、インタプリタやオペレーティングシステムなどのプログラム実行環境、例えばJavaプログラム実行環境において、アプリケーション単位あるいはスレッド単位で「トレースあり」と「トレースなし」との切換えによるトレース実行を行えるようにしたプログラム実行方式及び当該プログラム実行方式を格納した記憶媒体に関する。

【0002】

【背景技術】 従来のインタプリタやオペレーティングシステムなどのプログラム実行環境において、システム開発時もしくはシステム運用時に、アプリケーションプログラムのバグを修正し、正常に動作させるためアプリケーションプログラムをデバッグ (debug) する手段として、ソースプログラムの中にデバッグプリントを挿入することが多い。

【0003】 この方法でバグを含んだソースプログラムのデバッグ (バグ修正) を行うためには、図1に示すように、正式用のソースプログラム2にデバッグプリントを挿入してデバッグ用のソースプログラム2aに書き直し、デバッグ用のソースプログラム2aをコンパイルして実行ファイル3aを作成し、プログラム実行環境1でデバッグ用の実行ファイル3aを実行することによってバグ取りを行う。そして、デバッグ用のソースプログラム2aのバグ修正が完了したら、正式用のソースプログラム2も同様に修正し、実行ファイル3にコンパイルする。

【0004】 しかし、デバッグプリントを挿入する方法では、図1に示すように、正式用とデバッグ用の最低2種類のソースプログラム2、2aが混在することになるので、同一のソースプログラムについて2種類以上のプログラムを記述しなければならず、バグ修正作業が非常に乱雑となり、コード解析にも支障をきたすことがある。

【0005】 また、ソースプログラムにデバッグプリントを挿入することによってデバッグする方法では、デバッグ出力が待ち状態になる可能性があるため、デバッグプリントを入れることで処理のタイミングが変わる可能性がある。例えば、図2(a)に示すように、正式用のソースプログラム2が、スレッド (Thread; プログラム実行管理単位) AとスレッドBからなり、スレッドAが処理SA1、SA2、SA3によって、スレッドBが処理SB1、SB2、SB3によって構成されており、スレッドAの処理SA1、SA2、SA3が順次実行された後、スレッドBの処理に制御が移り、処理SB1、SB2、SB3が順次実行されるとする。このとき、図2(b)に示すように、デバッグ用のソースプログラム2aにおいて、スレッドAの処理SA2と処理SA3との間にデバッグプリントが挿入されたとすると、スレッドAにおける処理SA1、SA2に続いてデバッグプリントの処理が実行される。ここで、デバッグ出力が待ち状態になってしまったりすると、制御がスレッドBへ移り、スレッドBで処理SB1、SB2、SB3が順次実行された後、再び制御がスレッドAに戻って処理SA3が実行され、正式用のソースプログラム2と実行順序が変わってしまうような不都合が生じる恐れがある。

【0006】 さらに、デバッグプリントを挿入することによってデバッグする方法では、ある現象を把握するためのタイミングのコツを知る必要があり、開発者にとって効率的に作業が行えないという問題点があり、デバッグに要する時間が非常に長いものとなり、非効率的であった。

【0007】 また、デバッグプリントを挿入する以外のデバッグ手段としては、デバッガを利用する方法もある。しかし、デバッガを用いる方法では、デバッガを使用するための知識が必要となり、また組み込み機器のためのソフトウェアであれば、ハードウェア毎に設定が異なり、対象とするハードウェアに関する知識が必要となるため、デバッガを習熟するための時間やデバッグに至るまでの時間が長く掛かる不都合がある。

【0008】 このデバッガの問題を解決するための公知技術としては、インタプリタ言語であるBASICにトレース機能が存在し、この仕組みを取り入れることが考えられる。しかし、BASICのトレース機能は、ソースプログラムにトレース開始/終了のステートメントを挿入することで実現できるため、図3(a)(b)に示すように、デバッグ用のソースプログラム2aでは、正式用のソースプログラム2に対してトレースの開始位置にトレースONを、トレースの終了位置にトレースOFFをそれぞれ挿入する必要がある。よって、結局ソースプログラムを書き直す必要があり、デバッグプリントを挿入するのと同様、作業時間と手間が掛かってしまうことになる。

【0009】

【発明の開示】本発明の目的とするところは、プログラム実行環境において、アプリケーションプログラムを容易かつ作業効率よくデバッグできるプログラム実行方式を提供することにある。また、そのプログラム実行方式を格納した記憶媒体を提供することにある。

【0010】本発明にかかるプログラム実行方式は、トレース機能を有する環境でプログラムを実行することができるプログラム実行環境と、トレース機能を有しない環境で実行することができるプログラム実行環境とを備え、前記両プログラム実行環境が選択的に実行可能となっていることを特徴としている。

【0011】また、本発明にかかる記憶媒体は、トレース機能を有する環境でプログラムを実行することができるプログラム実行環境と、トレース機能を有しない環境で実行することができるプログラム実行環境とを備え、両プログラム実行環境が選択的に実行可能となっているプログラム実行方式を格納したものである。ここで記憶媒体としては、特にその種類は限定されることはなく、ハードディスク、CD類、DVD、MO、各種ICメモリなどが該当する。

【0012】本発明にあつては、複数のプログラム実行環境が切り換え可能となっており、トレース機能を有するプログラム実行環境とトレース機能を有しないプログラム実行環境のいずれかでプログラムを実行可能となっているため、プログラムのソースファイルにデバッグプリント等のトレースのための仕組みが必要なくなる。従って、プログラムのデバッグを行うため、そのソースプログラムを書き直したり、デバッグプリントを挿入したりする必要がなくなるので、あるいはソースプログラムの書き替え作業をわずかなものに行うことができるので、プログラムを容易かつ作業効率よくデバッグすることができる。よって、インタプリタやアプリケーションプログラムなどのプログラム実行環境において、システム開発時やシステム運用時に、デバッグのための作業時間を削減でき、また、ソースプログラムを書き直す必要がないので、デバッグ時のコード解析も簡単になる。

【0013】また、本発明の方式によれば、アプリケーションプログラムの実行中でもトレース環境が存在するため、トレース実行の切換え自由度が高くなる。例えば、アプリケーションプログラム起動前における切換え、アプリケーションプログラム実行中での切換え、アプリケーションプログラム内のスレッド毎の切換え等が可能になる。

【0014】

【発明の実施の形態】（第1の実施形態）プログラム実行環境としてJavaプログラム実行機構（Java仮想マシン）の場合を例として、以下本発明の一実施形態を説明する。従来のJava仮想マシンでは、単一のプログラム実行環境によってスレッドを実行しているが、本発明にかかるJava仮想マシンは、複数のプログラ

ミング実行環境を有している。

【0015】図4はJava仮想マシン4を用いた本発明にかかるJavaプログラム実行環境の概要を表している。この実施形態では、1つのJava仮想マシン4（Virtual Machine；プログラムから生成されたバイトコード、すなわち中間ファイルを実行するための環境）に対し、バイトコードを1命令ずつ逐次解釈しながら実行するためのインタプリタ5、6を2つ備え、両インタプリタ5、6で共有するデータ領域7を有している。一方のインタプリタ5（以下、通常インタプリタという。）はトレース機能を備えておらず、通常のインタプリタ形式でスレッドを解釈して実行する。他方のインタプリタ6（以下、トレース用インタプリタという。）はトレース機能を備えており、プログラムを1ラインや1命令ずつ実行し、トレースを行ってメモリの内容を外部に出力することができる。トレース結果の出力は、RS-232Cのようなパラレル出力やシリアル出力から出力させることができる。トレースの出力単位は、バイトコード単位、メソッド（関数）呼び出し単位のどちらでも出力可能である。

【0016】このJava仮想マシン4においては、両インタプリタ5、6を選択的に切り換えてスレッドを実行することができる。インタプリタ5、6の切換え方法ないし切換え時期としては種々用意しておくことができる。ここでは、例として、(1)アプリケーションプログラムが起動する前におけるインタプリタの切換え機能、(2)アプリケーションプログラム実行時におけるインタプリタの切換え機能、(3)スレッド毎のインタプリタ切換え機能、(4)局所的なトレースの切換え機能について説明する。Java仮想マシン4は、これらのインタプリタ又はトレースの切換え機能のうち1つだけを備えていてもよく、複数備えていてもよい。

【0017】まず、アプリケーションプログラム起動前のインタプリタ切換えについて説明する。このJava仮想マシン4には、スレッドを管理するための構造体が存在しており、そのメンバーとしてインタプリタ切換えのためのフラグが用意されている。そして、アプリケーションプログラム起動時には、当該フラグによって、2つのインタプリタのうち通常インタプリタ5で実行するか、トレース用インタプリタ6で実行するかが決まる。一方、Java仮想マシン4またはアプリケーションプログラムには、トレース有り、トレース無しのいずれかを設定するための環境変数（インタプリタを実行するための変数）を設けてあり、コマンド入力等によって環境変数をトレース有り、トレース無しのいずれかに設定できるようになっている。

【0018】しかして、このプログラム実行環境において、コマンド入力等によって環境変数をトレース有りに設定してアプリケーションプログラムを起動すると、図5のフロー図に示すように、環境変数によって上記フラ

5

グがトレース有りに設定され（ステップS1でトレース設定オン）、アプリケーションプログラムで使用されるスレッド全てに対応づけられる。従って、プログラム実行環境がトレース用インタプリタ6に設定され（ステップS2）、いずれのスレッドもトレース用インタプリタ6で実行され、トレース結果が外部へ出力される（ステップS3）。

【0019】また、コマンド入力等によって環境変数をトレース無しに設定してアプリケーションプログラムを起動すると、環境変数によって上記フラグがトレース無しに設定され（図5のステップS1でトレース設定OFF）、アプリケーションプログラムで使用されるスレッド全てに対応づけられる。よって、プログラム実行環境が通常インタプリタ5に設定され（ステップS4）、いずれのスレッドもトレース機能無しの通常インタプリタ5で実行される（ステップS5）。

【0020】このようにして2つのインタプリタ5、6を切替えるようにすれば、Java仮想マシン4が動作する実行環境でトレースの有無を設定できるので、デバッグを行なうためにソースプログラムを変更する必要がなく、アプリケーションプログラムを容易かつ作業効率よくデバッグできる。

【0021】次に、アプリケーションプログラム実行時におけるインタプリタ切換え機能について説明する。アプリケーションプログラムの実行時（起動後）にインタプリタを切換える機能を持たせる場合には、トレース有り、トレース無しのいずれかを設定するための環境変数（インタプリタを実行するための変数）を設け、インタプリタ実行中に、コマンド入力等によって環境変数をトレース有り、トレース無しのいずれかに再設定できるようにする。この環境変数は、初期値ではトレース無しに設定されている。

【0022】しかし、アプリケーションプログラムが起動されると、図6のフロー図に示すように、環境変数がトレース無しに設定されているので、通常インタプリタ5でスレッドを実行可能な状態で起動する（ステップS11）。このときには、まだコマンド入力はないので（ステップS13でオフの場合）、トレース機能がオフの通常インタプリタ5が作動し（ステップS14）、トレース無しでスレッドが実行される（ステップS15）。

【0023】アプリケーションプログラムが終了していなければ（ステップS18でNOの場合）、ステップS13～S18の処理が繰り返されるが、実行途中でコマンド入力（ステップS12）によってトレース機能がオンとなるように環境変数が書き替えられた場合（ステップS13でオンの場合）には、プログラム実行環境がトレース用インタプリタに切り替えられ（ステップS16）、トレース用インタプリタ6で各スレッドを実行しながらトレースが実行され、トレース結果が外部へ出力

6

される（ステップS17）。

【0024】また、トレース機能がオンになった状態でスレッドが実行されているときに、コマンド入力（ステップS12）によってトレース機能がオフに切り替えられた場合（ステップS13でオフの場合）には、再びプログラム実行環境が通常インタプリタに切り替えられ（ステップS14）、トレース機能のない通常インタプリタ5でスレッドが実行される（ステップS15）。

【0025】このようにしてアプリケーションプログラムの実行中に2つのインタプリタを切換えられるようにしても、Javaが動作する実行環境でトレースの有無を設定できるので、デバッグを行なうためにソースプログラムを変更する必要がなく、アプリケーションプログラムを容易かつ作業効率よくデバッグできる。

【0026】次に、スレッド毎のインタプリタ切換え機能について説明する。アプリケーションプログラムに存在するスレッド個々に対して、それぞれトレースを実行するか否かを設定できるようにする場合には、トレース用インタプリタ6をオンにする機能を組み込まれたトレース専用のスレッドクラス8を提供しておく。そして、図7のスレッドCのように、トレースを行わないスレッドに対しては、当該トレース用スレッドクラス8（または、当該トレース用クラス8を継承しているスレッドクラス）を継承させることなく通常のスレッドクラスを継承してスレッドを生成させる。一方、図7のスレッドA、Bのように、トレースを行いたいスレッドに対しては、トレース用スレッドクラス8（または、トレース用スレッドクラス8を継承したスレッドクラス）を継承させておけば、実行時にトレース用インタプリタ6が実行されるよう、スレッド生成時に内部でスレッド切換えの設定が行われる。しかし、トレース用スレッドクラス8を継承していないスレッドCが実行される場合には、Java仮想マシン4が通常インタプリタ5に設定され、通常インタプリタ5によって当該スレッドが実行され、トレースは行われぬ。これに対し、トレース用スレッドクラス8を継承しているスレッドA、Bが実行された場合には、当該スレッドA、Bはトレース用インタプリタ6によって実行され、トレース結果が出力される。

【0027】スレッド毎のトレース切換えを行うことができるようにした場合の具体的なスレッド実行手順は、図8に示すようになる。すなわち、アプリケーションプログラム起動前に環境変数がトレース無しに設定されている場合には、トレース設定の有無を判別することによって（ステップS21）トレース設定OFFと判断され、トレース用スレッドクラス8を継承しているか否かにかかわらずJava仮想マシン4は通常インタプリタ5に設定され（ステップS23）、トレース無しでスレッドが実行される（ステップS24）。

【0028】また、アプリケーションプログラム起動前

10

20

30

40

50

に環境変数がトレース有りに設定されている場合には、トレース設定の有無を判別することによって（ステップS21）トレース設定ONと判断された後、さらに、実行しようとするスレッドがトレース用スレッドクラス8を継承しているか否かを判別される（ステップS22）。このとき当該スレッドがトレース用スレッドクラス8を継承していない場合には、Java仮想マシン4は通常インタプリタ5に設定され（ステップS23）、トレースなしでスレッドが実行される（ステップS24）。これに対し、スレッドがトレース用スレッドクラス8を継承している場合には、Java仮想マシン4はトレース用インタプリタ6に設定され（ステップS25）、当該スレッドはトレース用インタプリタ6で解釈実行され、トレース出力が外部へ出力される（ステップS26）。

【0029】このような方法では、スレッドの継承元をトレース用のスレッドクラスと、トレース用インタプリタ6をオンにする機能のない通常のスレッドクラスとに変更するだけでよく、ソースプログラム自体の変更作業を最小の手間にする必要がないので、アプリケーションプログラムを容易かつ作業効率よくデバッグできる。あるいは、スレッドの継承元のスレッドクラスにトレース有りとトレース無しを切り替えるためのプロパティ等を組み込んでおき、コマンドによって切り替えられるようにしてもよい。

【0030】次に、局所的なトレースの切換え機能、すなわちアプリケーションプログラムやスレッドの一部分だけのトレース切換え機能について説明する。この場合には、トレース専用のスレッドクラスを提供しておき、そのスレッドクラスもしくは当該スレッドクラスを継承したスレッドクラスでは、トレース用インタプリタ6をトレース実行モードとトレース中止モードとに切り替えるためのメソッド（関数）を利用できるようにする。

【0031】図9は局所的なトレースの切り替えについて説明する図であって、スレッドA、B、Cのソースプログラム9a、9b、9cに含まれている”トレースON”は、トレース用インタプリタ6をトレース実行モードに切り替えるためのメソッド、”トレースOFF”は、トレース用インタプリタ6をトレース中止モードに切り替えるためのメソッドであって、いずれもメソッド呼び出し時に内部で環境変数の設定を行うことにより、トレース用スレッド6をトレースを実行する状態とトレースを実行しない状態とに設定するものである。また、トレース有りのインタプリタ6は、初期状態がトレース中止モードで実行されるとする。

【0032】しかして、これらのソースプログラム9a、9b、9cをコンパイルした実行ファイルが、実行される場合には、図10に示すフロー図に従う。すなわち、図9のスレッドA、Bの場合のように、コマンド入力等によって環境変数をトレース有りに設定してアプリケーションプログラムを起動すると（ステップS31で

トレース設定オン）、環境変数の設定によってトレース用インタプリタ6が実行される（ステップS32）。トレース用インタプリタ6が実行されると、トレース用フラグは初期値0に設定され（ステップS33）、トレースメソッド（”トレースON”、”トレースOFF”）が通常の処理コマンドから判別する（ステップS34）。このトレース用フラグは、トレース用インタプリタ6をトレース実行モードとトレース中止モードに設定するものであって、トレース用フラグが0の場合にはトレース中止モードに設定され、1の場合にはトレース実行モードに設定される。

【0033】トレース用フラグの初期値は0となっているので、スレッドA、Bの処理SA1、SB1は、トレース中止モードでスレッドが実行されるが（ステップS34、S38、S40）、メソッドが見つかり（ステップS34）、そのメソッドが”トレースON”か”トレースOFF”か判別し（ステップS35）、”トレースON”であればトレースフラグを1に設定して次の処理に進み（ステップS36）、”トレースOFF”であれば、トレースフラグを0に設定して次の処理に進む（ステップS37）。

【0034】このとき”トレースON”のメソッドであれば、トレースフラグは1であると判別される（ステップS38）ので、次の処理SA2、SA3；SB2、SB3はトレース実行モードとなり、トレース用インタプリタ6によりスレッドを実行されながらトレースが実行される（ステップS39）。

【0035】ついて、”トレースOFF”のメソッドが実行されると、トレースフラグが0に設定される（ステップS34、S35、S37）ので、1であると判別される（ステップS38）ので、トレース中止モードに戻り、次の処理SA4、SB4はトレース用インタプリタ6によりスレッドのみ実行される（ステップS40）。そして、すべての処理が実行されると、スレッドA、Bの実行が終了する（ステップS41）。

【0036】次に、図9のスレッドCの場合のように、コマンド入力等によって環境変数をトレース無しに設定してアプリケーションプログラムを起動すると（ステップS31でトレース設定オフ）、環境変数の設定によって通常インタプリタ5が起動される（ステップS42）。そして、スレッドCがトレース用のメソッドを含んでいるか否かにかかわらず、全ての処理SC1～SC4が通常インタプリタ5により実行される（ステップS43、S44）。

【0037】この場合には、スレッドにトレース用のメソッドを含んでいても通常インタプリタを実行することにより通常のスレッドのように実行することができるので、正式用のソースプログラムに当初からトレース用のメソッドを挿入しておけば、正式用のソースプログラムとデバッグ用のソースプログラムを別個に作成する必要

10

20

30

40

50

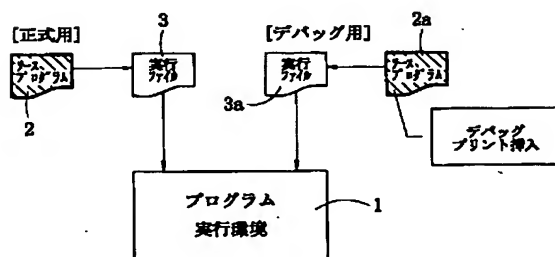
が無くなり、アプリケーションプログラムを容易かつ作業効率よくデバッグできる。

【0038】本発明にあっては、上記のようにトレース有りのプログラム実行環境とトレース無しのプログラム実行環境とを備えているので、図11に示すように、ソースプログラム2をコンパイルして実行ファイル3を作成した後、トレース無しのプログラム実行環境で実行ファイル3（スレッド）を実行させることができ、またトレース有りのプログラム実行環境で実行ファイル3をデバッグすることができる。よって、従来例のようにデバッグプリントを挿入されたソースプログラム2aを別途作成してデバッグ用の実行ファイル3aを作成する必要が無くなる。このため、本発明によれば、デバッグのための作業時間を大幅に短縮することができる。例えば、デバッグプリント挿入によってデバッグを行う方法では、図12に示すように、正式用のソースファイルにデバッグプリントを挿入するのに要する時間が1ファイルあたり180秒、1ファイルのコンパイル時間が10秒とし、全部で200ファイルとすると、デバッグ用の実行ファイルを作成するのに、約600分（約10時間）掛かることになる。これに対し、アプリケーションプログラム実行前にプログラム実行環境のトレース有り、無しを切り替える本発明方法では、正式用の実行ファイルをそのまま使用できるので、デバッグのための実行ファイルを作成する時間は必要ない。よって、本発明の方法によれば、デバッグ作業に要する手間と時間を大幅に軽減することができる。

【0039】

【発明の効果】本発明によれば、トレース機能を有するプログラム実行環境とトレース機能を有しないプログラム実行環境のいずれかでプログラムを実行可能となっているため、プログラムのソースファイルにデバッグプリント等のトレースのための仕組みが必要なので、プログラムのデバッグを行うためにソースプログラムを書き直したり、デバッグプリントを挿入したりする必要が無くなり、プログラムを容易かつ作業効率よくデバッグすることができるようになる。

【図1】



【図面の簡単な説明】

【図1】従来のデバッグ方法の一例を示す概略図である。

【図2】（a）は同上のデバッグ方法に用いられている正式用のソースプログラムを示す図、（b）はデバッグ用のソースプログラムを示す図である。

【図3】トレースによりデバッグを行う従来方法を説明する図であって、（a）は正式用のソースプログラムを示す図、（b）はデバッグ用のソースプログラムを示す図である。

【図4】本発明の一実施形態にかかるJava仮想マシンの構成を示す概略図である。

【図5】同上のJava仮想マシンにおいて、アプリケーションプログラム起動前にインタプリタを切り替える手順を示すフロー図である。

【図6】図4のJava仮想マシンにおいて、アプリケーションプログラム実行中にインタプリタを切替える手順を示すフロー図である。

【図7】図4のJava仮想マシンにおいて、スレッド毎にトレースの有無を切替える方法を説明する図である。

【図8】同上のJava仮想マシンにおいて、スレッド毎にインタプリタを切り替える手順を示すフロー図である。

【図9】図4のJava仮想マシンにおいて、局所的にトレースの有無を切り替える方法を説明する図である。

【図10】同上のJava仮想マシンにおいて、局所的なインタプリタの切替え手順を示すフロー図である。

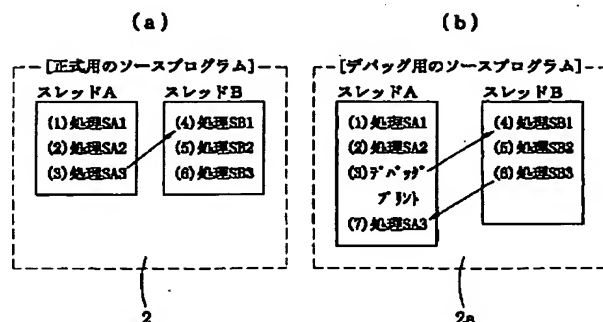
【図11】本発明のデバッグ方法と従来例のデバッグ方法を比較して説明する図である。

【図12】本発明のデバッグ方法に要する時間と従来例のデバッグ方法に要する時間を比較して示す図である。

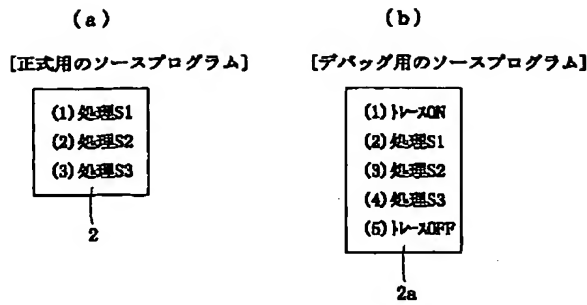
【符号の説明】

- 4 Java仮想マシン
- 5 通常インタプリタ
- 6 トレース用インタプリタ

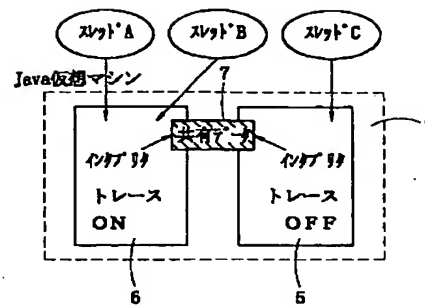
【図2】



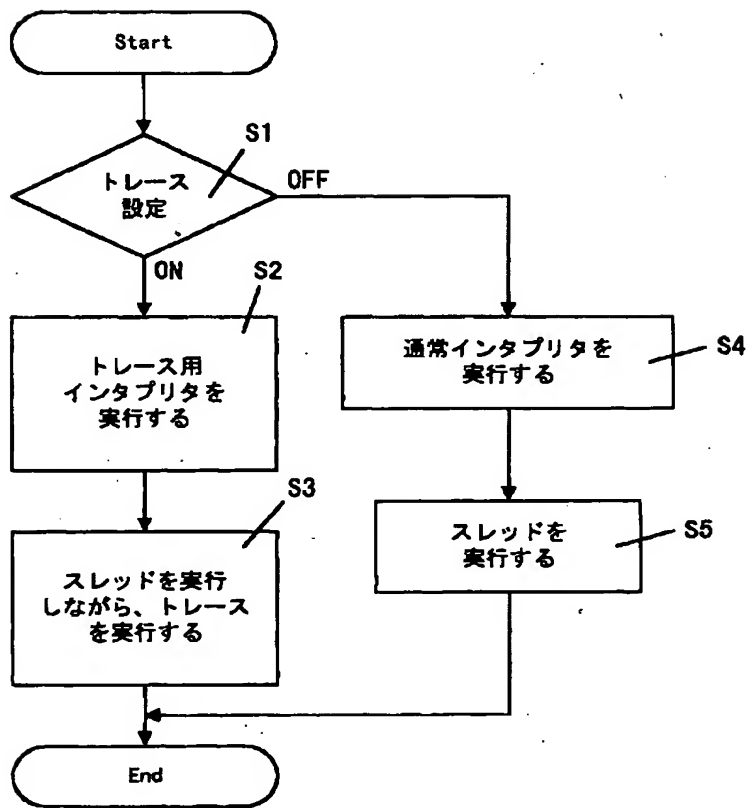
【図3】



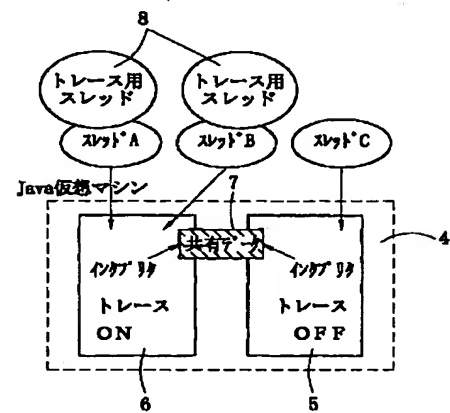
【図4】



【図5】



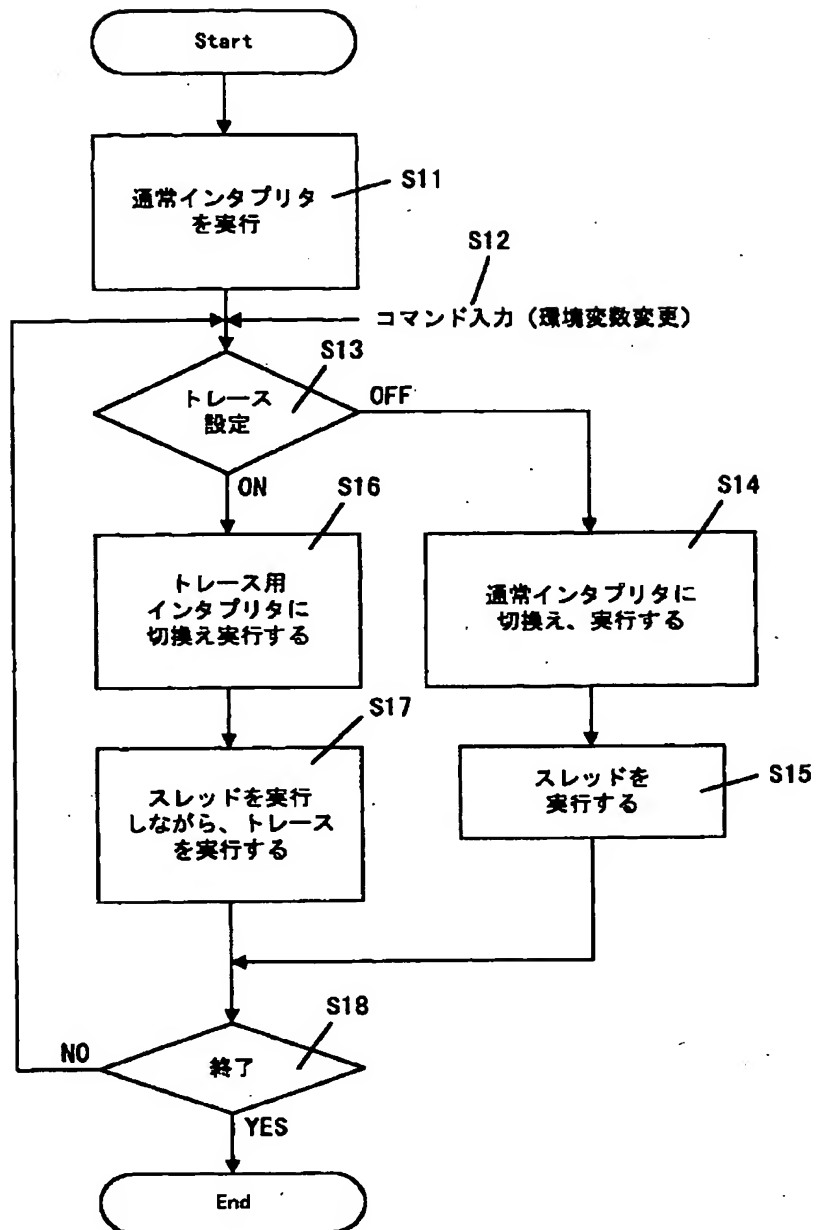
【図7】



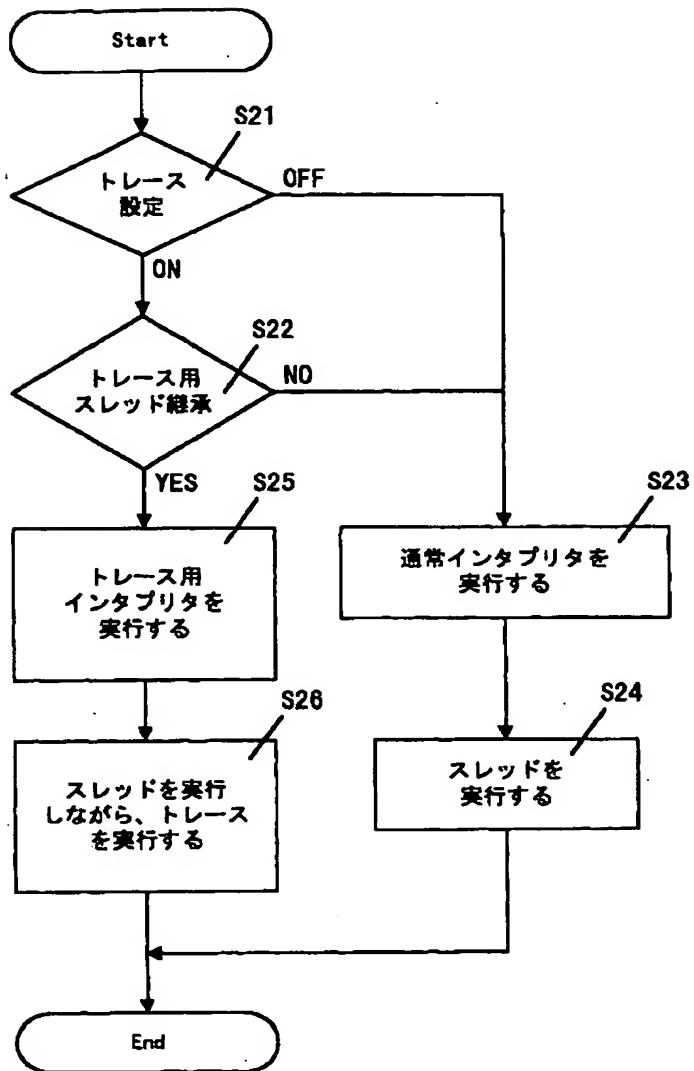
【図12】

	1ファイルあたりの デバッグソフトの挿入時間	コンパイル 時間	ファイル数	推定時間
従来	180秒	10秒	200	約500分
本発明	0秒	0秒	200	0分

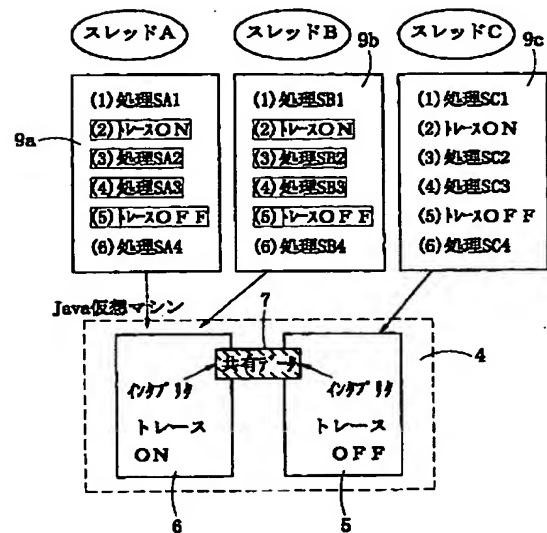
【図 6】



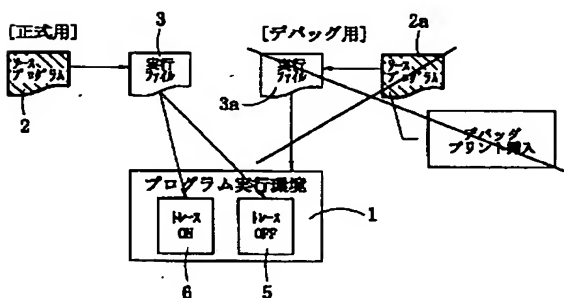
【図8】



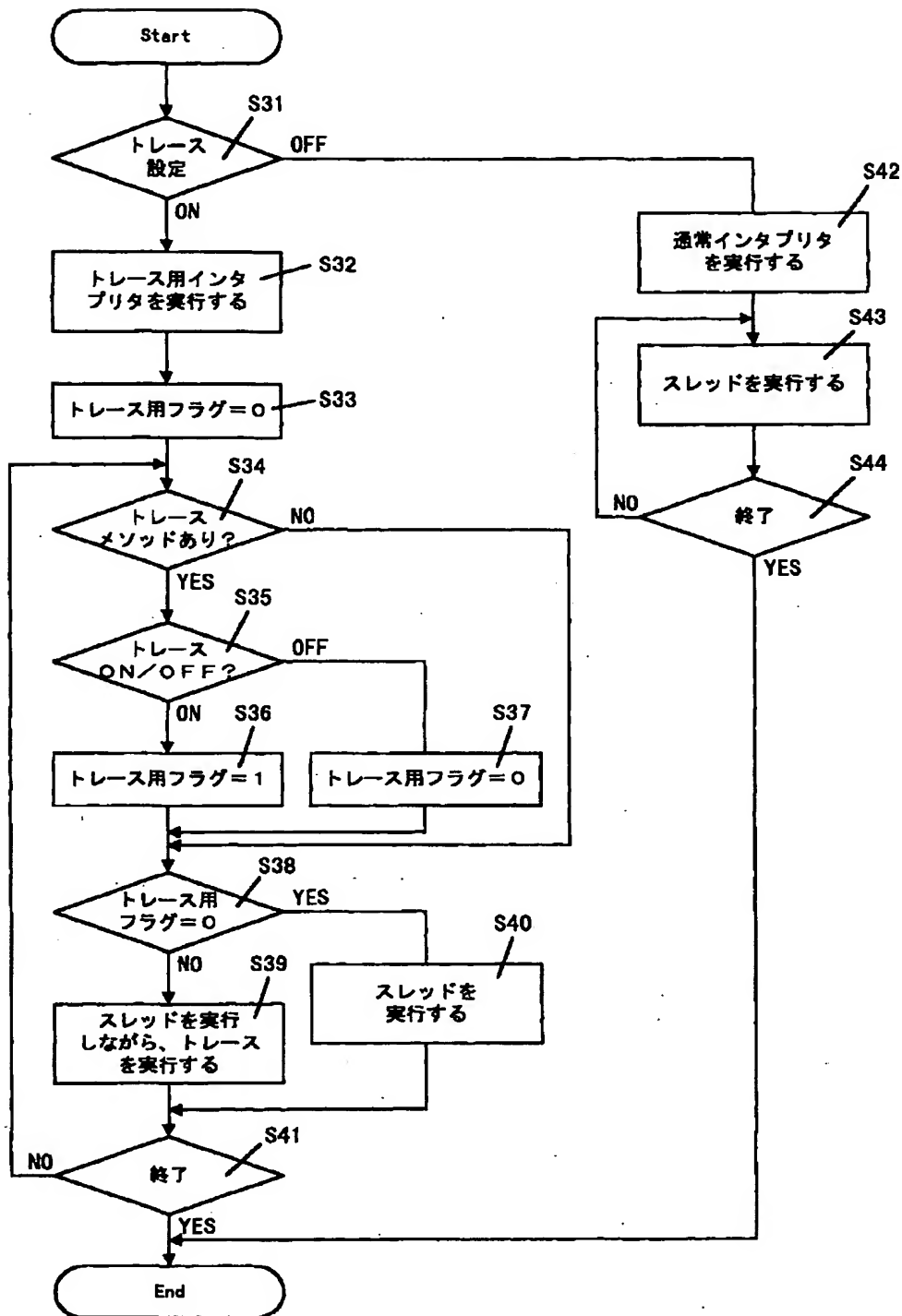
【図9】



【図11】



【図10】



フロントページの続き

(72)発明者 小中 義治
京都府京都市右京区花園土堂町10番地 オ
ムロン株式会社内
(72)発明者 門脇 正規
京都府京都市右京区花園土堂町10番地 オ
ムロン株式会社内

(72)発明者 村井 謙一
京都府京都市右京区花園土堂町10番地 オ
ムロン株式会社内
F ターム (参考) 5B042 GA08 GA23 HH07 HH30 MA10
MA20